

FATFS LLD SDS

TABLE OF CONTENTS

1	INTRODUCTION	1
2	KEY FEATURES	1
3	HARDWARE SUPPORT (EVM/SOCS)	1
4	DESIGN OVERVIEW	1
4.1	PERIPHERAL DEVICE DRIVER	2
4.2	DEVICE SPECIFIC MODULE LAYER.....	2
4.3	APPLICATION CODE.....	2
4.4	OSAL	2
4.5	CSL REGISTER LAYER	2
6	DRIVER CONFIGURATION	3
6.1	BOARD SPECIFIC CONFIGURATION	3
6.2	FATFS CONFIGURATION STRUCTURE	3
6.3	APIS.....	3
6.4	USAGE.....	3
6.5	API CALLING SEQUENCE.....	3
6.6	FLOW CHART	4
7	EXAMPLES	5
7.1	EEPROM READ:	5
7.1.1	<i>Building the examples:</i>	5
7.1.2	<i>Running the examples</i>	5
7.1.3	<i>Supported platforms:</i>	6
8	TEST	6
8.1	BUILDING THE EXAMPLES:.....	6
8.2	RUNNING THE EXAMPLES	7
8.3	SUPPORTED PLATFORMS:.....	7
9	MIGRATION GUIDE	7
10	BENCHMARKING	7

1 Introduction

The FATFS module provides an interface to configure a driver for FAT file system compatible device that connects via the MMCSD, etc.

2 Key Features

- Configures FATFS for disk operations
 - Disk Initialize
 - Disk Read
 - Disk Write

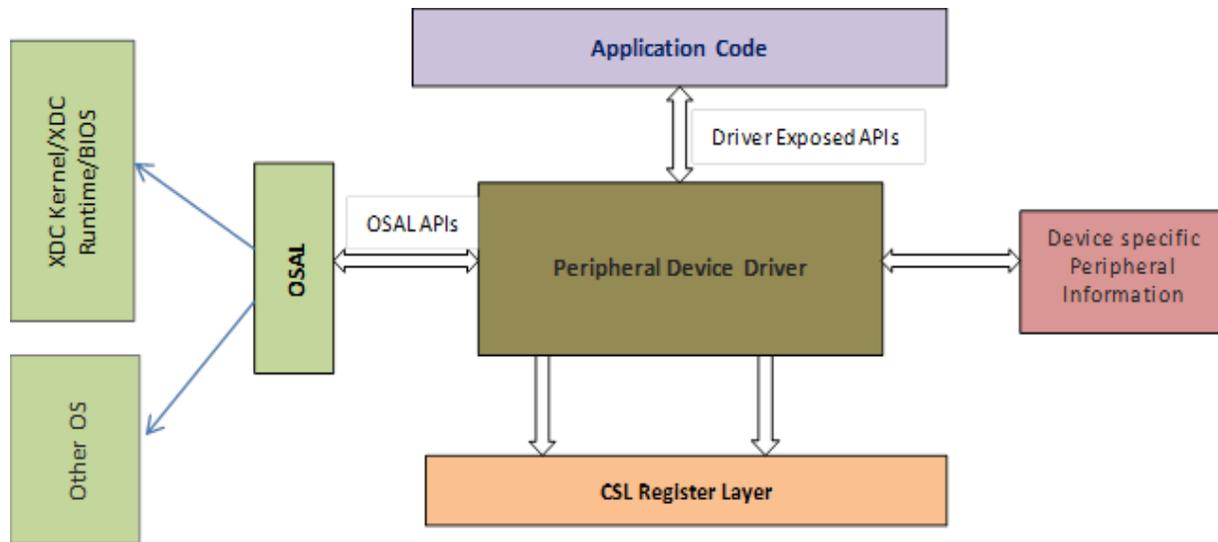
3 Hardware Support (EVM/SoCs)

Board	SoC	Cores
AM572x IDK EVM	AM572x	A15 & C66X
AM572x GP EVM	AM572x	A15 & C66X
AM571x IDK EVM	AM571x	A15 & C66X

4 Design Overview

The FATFS driver provides a well-defined API layer which allows applications to use the FATFS peripheral to send and receive data.

The below figure which shows the FATFS Driver architecture.



The figure illustrates the following key components:-

4.1 Peripheral device driver

This is the core FATFS device driver. The device driver exposes a set of well-defined APIs which are used by the application layer. The driver also exposes a set of well-defined OS abstraction APIs which will ensure that the driver is OS independent and portable. The driver uses the CSL register layer for MMR accesses.

4.2 Device specific module layer

This layer implements a well-defined interface which allows the core FATFS device driver to be ported to any device which has the same FATFS IP block. This layer may change for every device.

4.3 Application Code

This is the user of the driver and its interface through the well-defined APIs set. Application uses the driver APIs to send and receive data via the FATFS peripheral.

4.4 OSAL

The driver is OS independent and exposes all the operating system callouts via this OSAL layer.

4.5 CSL Register Layer

The FATFS driver uses the CSL functional layer to program the device IP by accessing the MMR (Memory Mapped Registers).

5 Driver Configuration

5.1 Application Specific Configuration

All the board specific configurations like enabling the clock and pin-mux of FATFS pins should be performed before calling any of the driver APIs. Once the board specific configuration is done then the driver API `FATFS_init()` should be called to initialize the FATFS driver.

5.2 FATFS Configuration Structure

The `FATFS_soc.c` file contains the declaration of the `FATFS_config` structure. This structure must be provided to the FATFS driver. It must be initialized before the `FATFS_init()` function is called and cannot be changed afterwards. For details about the individual fields of this structure, see the Doxygen help by opening `\docs\doxygen\html\index.html`.

5.3 APIs

In order to use the FATFS module APIs, the `FATFS.h` header file should be included in an application as follows:

```
#include <ti/drv/FATFS/FATFS.h>
```

The following are the FATFS APIs:

- **FATFS_init()** initializes the FATFS module.
- **FATFS_Params_init()** initializes an `FATFS_Params` data structure. It defaults to Blocking mode.
- **FATFS_open()** initializes a given FATFS peripheral.
- **FATFS_close()** deinitializes a given FATFS peripheral.

5.4 Usage

The application needs to supply the following structures in order to set up the framework for the driver:

- **FATFS_Params** specifies the transfer mode and any callback function to be used.

5.5 API Calling Sequence

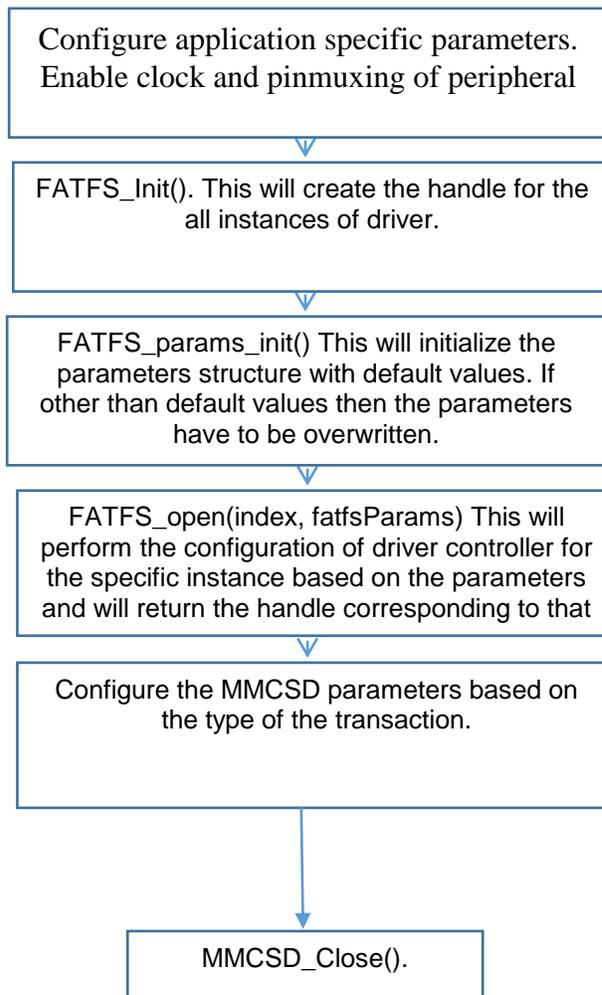
The below sequence indicates the calling sequence of FATFS driver APIs for a use case of write transaction in blocking mode:

```
FATFS_Handle FATFS;  
FATFS_Params FATFSParams;
```

```
FATFS_Params_init(&FATFSParams);  
  
FATFS = FATFS_open(peripheralNum, &FATFSParams);  
if (FATFS == NULL) {  
    /* Error opening FATFS */  
}
```

5.6 Flow chart

API flow path of FATFS driver is as shown in fig.



6 Examples

Following are the examples of supported for the FATFS Driver

6.1 Eeprom Read:

Each EVM will have ID memory (EEPROM), where board specific information like board Id and version information will be stored. This example will read the data from the EEPROM using the FATFS driver and verifies whether this data matches with the expected data. As a first step this will write the offset address of EEPROM from which data has to be read and then reads the number of bytes from the EEPROM.

6.1.1 Building the examples:

Following are list of FATFS projects which will reside the following location “packages/MyExampleProjects”

```
FATFS_BasicExample_AM571X_armExampleProject
FATFS_BasicExample_AM571X_c66xExampleProject
FATFS_BasicExample_AM572X_armExampleProject
FATFS_BasicExample_AM572X_c66xExampleProject
FATFS_BasicExample_AM572X_GpEvm_armExampleProject
FATFS_BasicExample_AM572X_GpEvm_c66xExampleProject
```

These projects have to be imported in CCS and have to be built. The “.out” files corresponding to each project will be generated after successfully compiling the projects.

Interrupt/Non-Interrupt Modes:

The example projects have to be recompiled to support interrupt and non-interrupt use cases

Following parameter have to be updated in FATFS_Soc.c file and the application have to be recompiled.

- Structure: FATFSInitCfg
- Parameter: enableIntr
 - True: interrupt
 - False: Non interrupt

6.1.2 Running the examples

The “.out” have to be loaded and executed. Then on the CCS console the result of the project execution will be displayed. If the project is executed successfully, then it will print “PASS” else will print “Data Mismatch”

6.1.3 Supported platforms:

AM572x GP EVM
AM572x IDK EVM
AM571x IDK EVM

7 Test

Each EVM will have ID memory (EEPROM), where board specific information like board Id and version information will be stored. This example will read the data from the EEPROM using the FATFS driver and verifies whether this data matches with the expected data. As a first step this will write the offset address of EEPROM from which data has to be read and then reads the number of bytes from the EEPROM.

This test application will test the use case for the following speeds
100 Kbps
400 Kbps

7.1 Building the examples:

Following are list of FATFS projects which will reside the following location
“packages/MyExampleProjects”

FATFS_BasicExample_AM571X_Idk_armTestProject
FATFS_BasicExample_AM571X_Idk_c66xTestProject
FATFS_BasicExample_AM572X_Idk_armTestProject
FATFS_BasicExample_AM572X_Idk_c66xTestProject
FATFS_BasicExample_AM572X_Evm_armTestProject
FATFS_BasicExample_AM572X_Evm_c66xTestProject

These projects have to be imported in CCS and have to be built. The “.out” files corresponding to each project will be generated after successfully compiling the projects.

Interrupt/Non-Interrupt Modes:

The example projects have to be recompiled to support interrupt and non-interrupt use cases

Following parameter have to be updated in FATFS_Soc.c file and the application have to be recompiled.

- Structure: FATFSInitCfg

- Parameter: enableIntr
 - True: interrupt
 - False: Non interrupt

7.2 Running the examples

The “.out” have to be loaded and executed. Then on the CCS console the result of the project execution will be displayed. If the project is executed successfully, then it will print “PASS” else will print “Data Mismatch”

7.3 Supported platforms:

AM572x GP EVM
AM572x IDK EVM
AM571x IDK EVM

8 Migration Guide

The driver supports multiple SoCs, Cores and different IP versions. Different IP versions are supported using function pointer based approach, whereas high lever driver APIs will remain same and these driver APIs will call the corresponding the correct version of IP specific implementation APIs using function pointers. This function pointer table will be fixed for each instance of the peripheral and will be defined in the main config structure, which resides in soc specific file “FATFS_soc.c”.

Users who are using the low level APIs(Device abstraction APIs: which perform hardware register read/write) have to use the high level APIs which are described in the section 6.3.

9 Benchmarking

Code size for library in bytes:

Initialized data : 40
Code: 5792